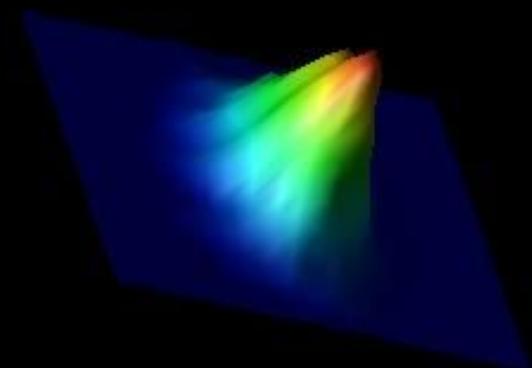
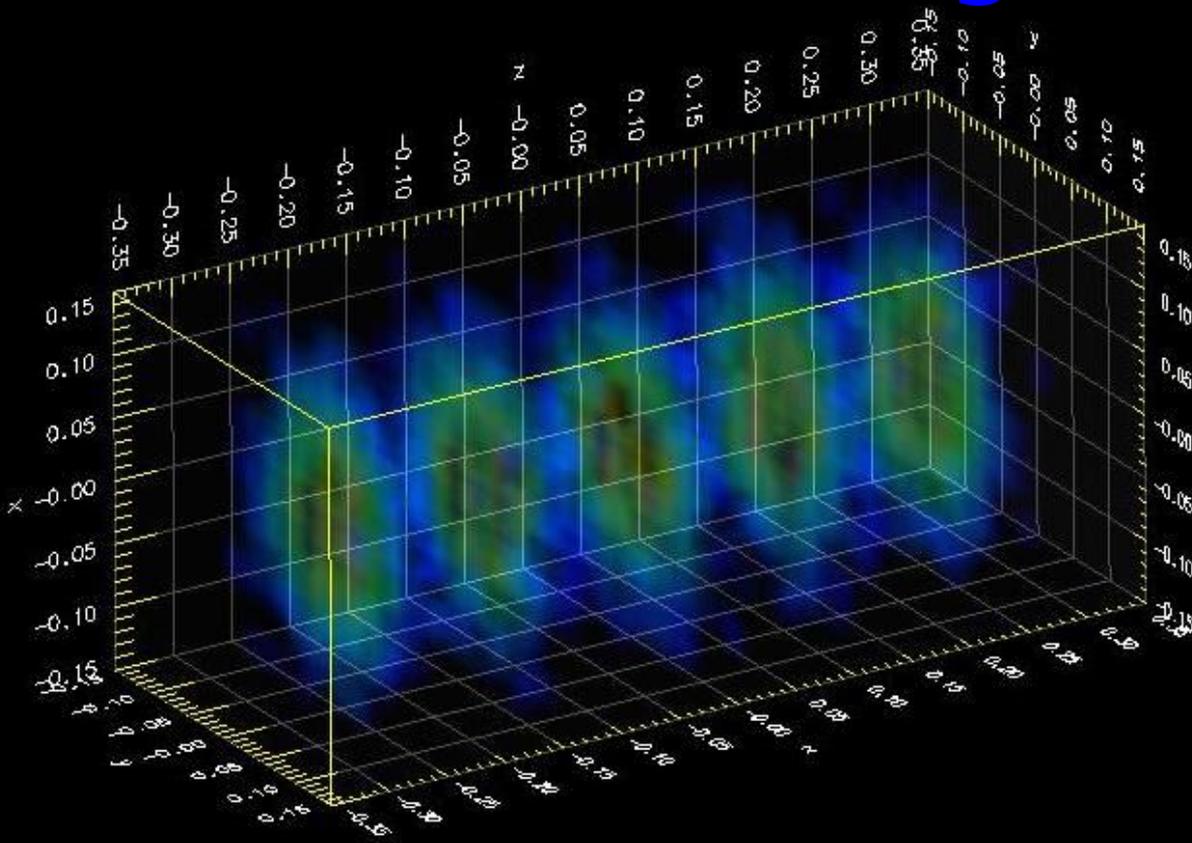


# Synergia

James Amundson and  
Panagiotis Spentzouris  
Fermilab



Horizontal Phase Space

# Synergia

---

- Problem Overview
- Synergia Overview
  - **IMPACT**
  - **mxyzptlk / beamline**
- Synergia
  - **combining codes**
  - **portable compilation**
  - **interface**
  - **documentation**
- Performance
  - **Scaling**
- Next Steps
- Visualization
  - **OpenDX**
  - **Demo**

# General Problem Overview: Particle Accelerator Modeling

---

- Typical problem: model behavior of
  - $O(10^{12})$  particles
  - Through 100's of elements
  - 100's-1000's-more revolutions in a circular accelerator
  - Each particle has six degrees of freedom  $(x, p_x, y, p_y, z, p_z)$

(Bad News)

# Single-Particle Dynamics

---

- In many situations, particle-particle interactions are negligible
  - Need only to track single particles
- Linear approximation
  - 6d-state transformation due to single step reduces to multiplication by 6x6 matrix (*map*)
  - Symplectic = Stable
    - Errors do not grow with number of steps

(Good News)

# Multi-Particle Dynamics

---

- Particle-particle interactions *are* important in many current problems
- Space Charge (*issue for Booster*)
  - Interaction of the beam with itself
    - *Our current interest*
- Others
  - Beam-beam (*issue for Tevatron*)
  - Electron cloud (*issue for LHC*)

(Bad News Returns)

# Multi-particle, cont.

---

- Use particle-in-cell (PIC) techniques
  - Macro particles
  - Solve continuous equations on discrete grid
- 65 x 65 x 65 grid typical size
- Need  $\sim 3,000,000$  particles in order to have an average of 10 particles per cell
- Parallel computers are necessary
  - See performance...

# More problems in particle accelerator modeling

---

- Accelerators are complicated devices
- Simulations have complicated inputs, complicated running conditions
- Analysis of simulations come in many forms
  - Multi-particle analyses are more complex than single-particle analyses

Problems are not simply numerical

# Tevatron lattice description

```

!
! constants
!
NBENDS := 774.0
BANGLE := TWOPT / NBENDS ! approx. 8.12 mrad
HANGLE = 3.87625450E-03 ! From N. Gelfand file
LAMBANGLE = 1.83577060E-03 ! From N. Gelfand file
CMAGBANGLE = 1.48790000E-03 ! Changed slightly to make
! 2*HANGLE + 3*LAMBANGLE + 2*CMAGANGLE = 2*BANGLE = 0.016235621
DOGANGLE = 3.272764E-3

LBFIELD = 6.1214
LHBFIELD = 2.921 ! half dipole length according to Norm G.
LLAMB = 5.521706 ! C0 abort lambertson
LCMAG = 3.73888 ! C0 C-magnet
LDOGBEND = 6.0706 ! 239 inches
LSEPTA = 3.5433

!
! -----
! main dipoles
! -----

! Next 4 are defined but never used. They are the magnetic fields in Tesla.
BENDFIELD = BANGLE*BRHO/LBFIELD
HBENDFIELD = HANGLE*BRHO/LHBFIELD
LAMBFIELD = LAMBANGLE*BRHO/LLAMB
CMAGFIELD = CMAGBANGLE*BRHO/LCMAG

BEND: SBEND, L = LBFIELD, ANGLE = BANGLE
BENDQ: MULTIPOLE, K1L = KBENDQ
HBEND: SBEND, L = LHBFIELD, ANGLE = HANGLE
LAMBEND: SBEND, L = LLAMB, ANGLE = LAMBANGLE
CMAGBEND: SBEND, L = LCMAG, ANGLE = CMAGBANGLE

! Next bends are added for fixed target
! They are the dogleg at D0 around the extraction septa and are on
! the Tevatron Main Bus.
! The distance between the upstream DOGPBEND, DOGBBEND is about 4 cm
! different from the distance between the downstream DOGPBEND, DOGBBEND.
! As a result, when these are on, the closure of the Tevatron is
! changed slightly. (This can only be seen in the results of a survey
! command.) The D0BUMPK1 and D0BUMPK2 are probably intended to
! be used as small adjustments to close this bump, but I have not
! ever used them for this. ppb
DOGPBEND: SBEND, L = LDOGBEND, ANGLE = DOGANGLE
DOGBBEND: SBEND, L = LDOGBEND, ANGLE = -DOGANGLE

! E0DOGP and E0DOGM are for the proton removal insert in E0.
! They have their own power supply, the Transrex that used to be used for
! the E0 Lambertsons.
! Most of the time these will be off. When they are turned on, they
! will bend the beams by DOGANGLE (see above)
! E0DOGBEND := 3.272764E-3
! Notice that E0DOGP has a kick of -E0DOGBEND.
! A positive kick is the same sign as the main bends.
! The E0 dogleg should displace the beam to the radial OUTSIDE.
! I've chosen to put them in as closed orbit correctors to make it
! easy to see how much they move the beam. (If I put them in as regular
! bends, I would have to do a survey and figure out how much it moves.)
! ppb 9/12/97
E0DOGBEND := 0.0
E0DOGP: HKICKER, L=LDOGBEND, KICK = -E0DOGBEND
E0DOGM: HKICKER, L=LDOGBEND, KICK = E0DOGBEND

```

■ ■ ■

```

E0DOGLEGM:LINE = ( DDOGEND1, E0DOGM, DDOGEND1 )
E0DOGLEGP:LINE = ( DDOGEND1, E0DOGP, DDOGEND1 )
E0COLL2: LINE = ( DR2COLLEND, ME02UHCL, ME02UVCL, DR2COLL2, E02HCL, &
E02VCL, DR2COLL2, ME02DHCL, ME02DVCL, DR2COLLEND )
E0COLL3: LINE = ( DR2COLLEND, ME03UHCL, ME03UVCL, DR2COLL2, E03HCL, &
E03VCL, DR2COLL2, ME03DHCL, ME03DVCL, DR2COLLEND )

E0DAMPK: LINE = ( DPHDAMPK, D3IN, DPODAMPK, D3IN, DPBHDAMPK, D3IN, DPBVDAMPK )
E0DAMPPU: LINE = ( DHDAMPPU, D3IN, DVDAMPPU )
LSTRE0DR2:LINE = ( E0DOGMP2, DDOGEND1, DE0SP6, E0DAMPK, DE0SP7, &
DBELL7A, E0COLL2, DBELL7A, E0COLL3, DBELL7A, &
E0DAMPPU, DE0SP8, DGV4, DE0SP9, DTAB, D3IN )

```

```

QUADE0D: LINE = ( DQUAD1END, HQUAD1F, DQUAD1END )
COLDBYP1: LINE = ( DCOLD1 )
STRAIGHTE0D: LINE = ( DE11END, HFWE11, DE11MID, VFWE11, DE11END )
COLDBYP2: LINE = ( DCOLD2 )
E0DOWN: LINE = ( ME0, LONGSTRE0D, QUADE0D, COLDBYP1, STRAIGHTE0D, COLDBYP2 )
E0DOWNR2: LINE = ( ME0, LSTRE0DR2, QUADE0D, COLDBYP1, STRAIGHTE0D, COLDBYP2 )
QUADE11: LINE = ( DBPMIN, HBPME11, DBPMOUT, HQUAD2D, DQOUT1 )
PACKE11: LINE = (DBPMIN1, VBPME11, DBPMOUT1, DHQUADC, TSQE0, HDE11, VDE11, &
DHQUADC, DPACKOUT1 )
DIPOLE: LINE = ( DBENDEND, BENDQ, BEND, BENDQ, DBENDEND )
E11: LINE = ( ME11, QUADE11, PACKE11, 4*DIPOLE )
QUADE12: LINE = ( DBPMIN, VBPME12, DBPMOUT, HQUAD3D, DQOUT )
PACKE12: LINE = ( DPACKIN, DHQUADC, TSX, TQX, VDE12, DHQUADC, &
DPACKU2D, DPACKOUT )
E12: LINE = ( ME12, QUADE12, PACKE12, 4*DIPOLE )
QUADE13: LINE = ( DBPMIN, HBPME13, DBPMOUT, HQUADF, DQOUT )
PACKE13: LINE = ( DPACKIN, TQFA4, TSF, HDE13, DPACKU2D, TSQ, DPACKOUT )
PACKE13R2:LINE = ( DPACKIN, TQFE1, TSF, HDE13, DPACKU2D, TSQ, DPACKOUT )

```

■ ■ ■

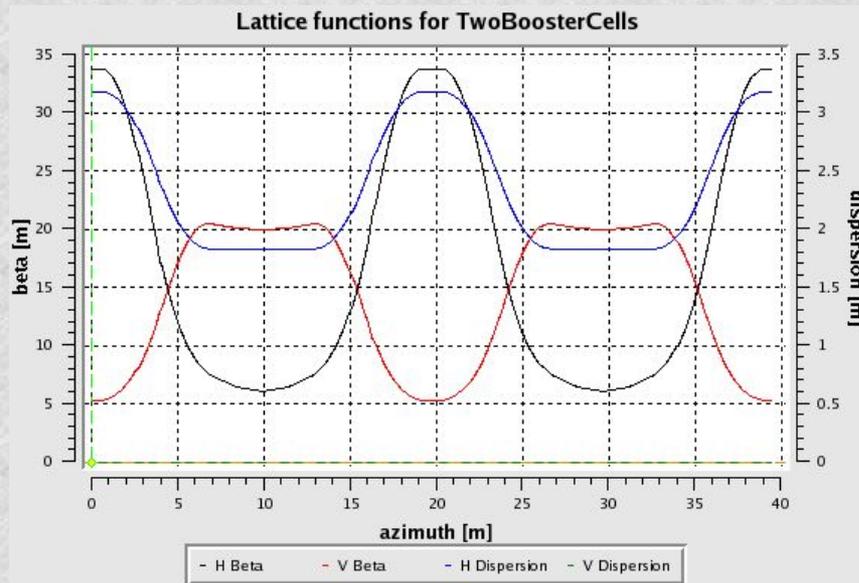
Full description is 3574 lines

■ ■ ■

# Existing software

- Mature single-particle tracking codes are available
- Some multi-particle codes available
  - Most written from scratch to address specific problems
    - “Just enough” single-particle code

Aesop by Leo Michelotti (FNAL)



We have decided to combine the best available existing codes

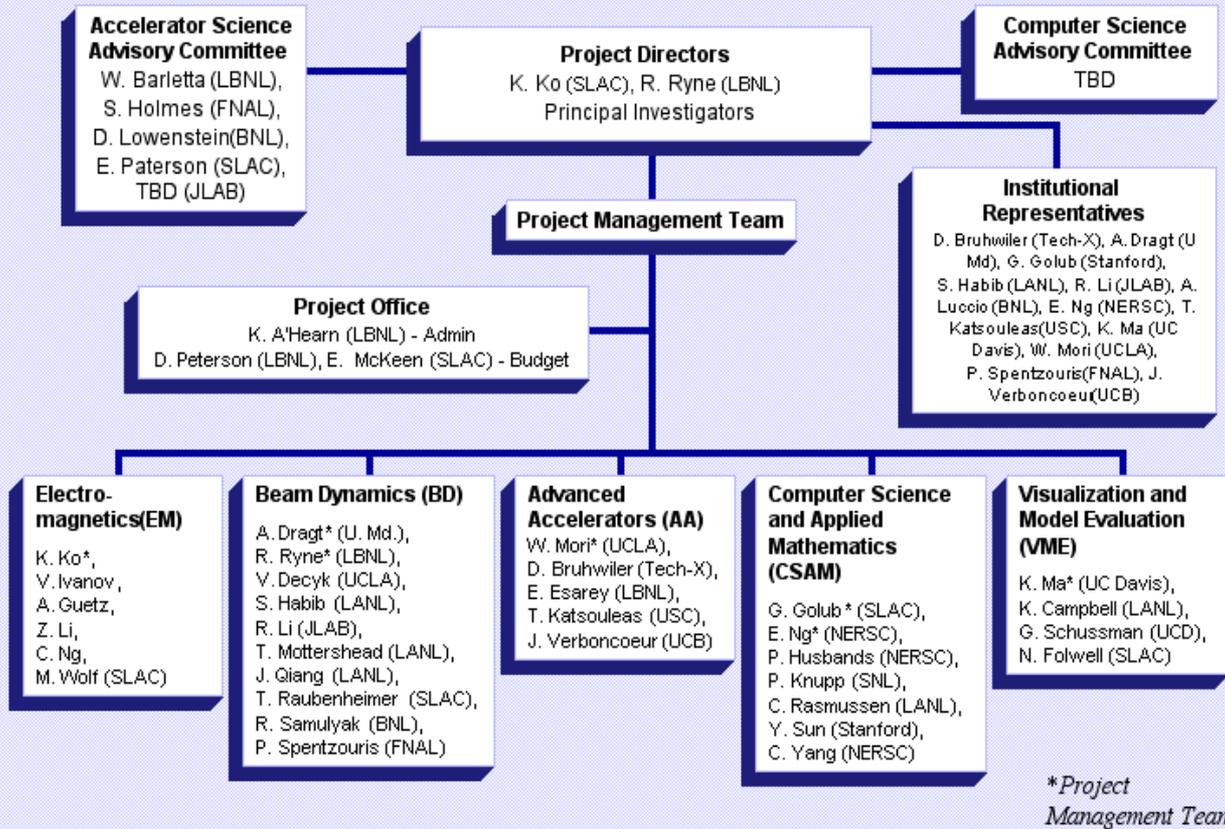
# The Synergia approach

---

- Take the best of existing codes
  - Single-particle
  - Multi-particle
- Create a framework
  - Extensible
- Solve practical problems in addition to numerical problems
- Project funded by SciDAC

# SciDAC Accelerator Modeling Project

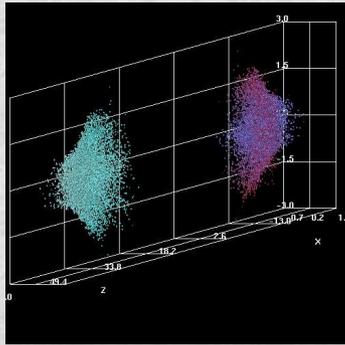
SciDAC Accelerator Modeling Project Org Chart  
August 2001



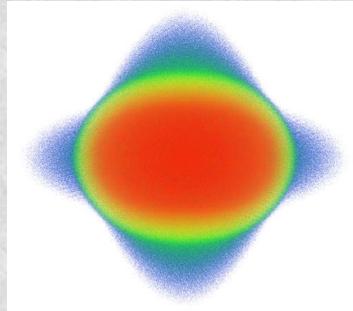
Members of a multi-institution collaboration. The charge : develop the next generation of parallel computing beam dynamics and accelerator modeling tools

Project funded by the SciDAC DOE program: \$3M in years; FNAL \$.35M

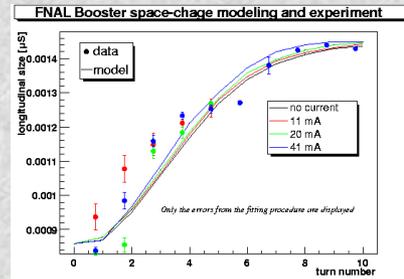
# Accelerator Modeling Collaboration



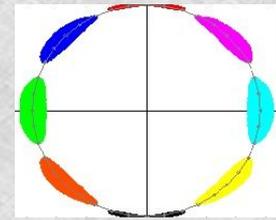
**LBL**  
Beam-beam modeling, space charge in linacs & rings, parallel Poisson solvers, collisions



**UC Davis**  
Visualization, multi-resolution techniques



**FNAL**  
Software Integration, Lie methods, space charge in rings, FNAL Booster sim/expt



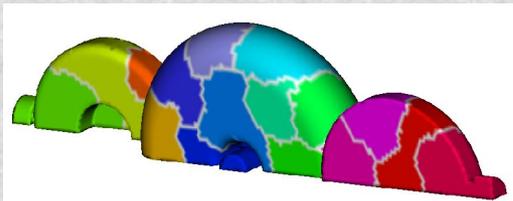
**BNL**  
Wakefield effects, Space charge in rings, BNL Booster simulation



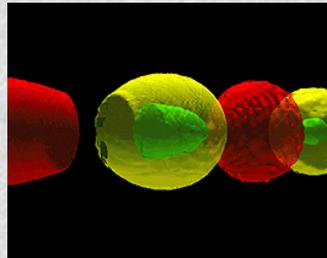
$$M = e^{if_2} e^{if_3} e^{if_4} \dots$$

$$N = A^{-1} M A$$

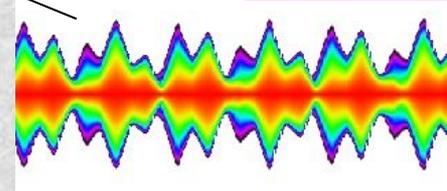
**U. Maryland**  
Lie Methods in Accelerator Physics, MaryLie



**SLAC**  
Electromagnetic component modeling

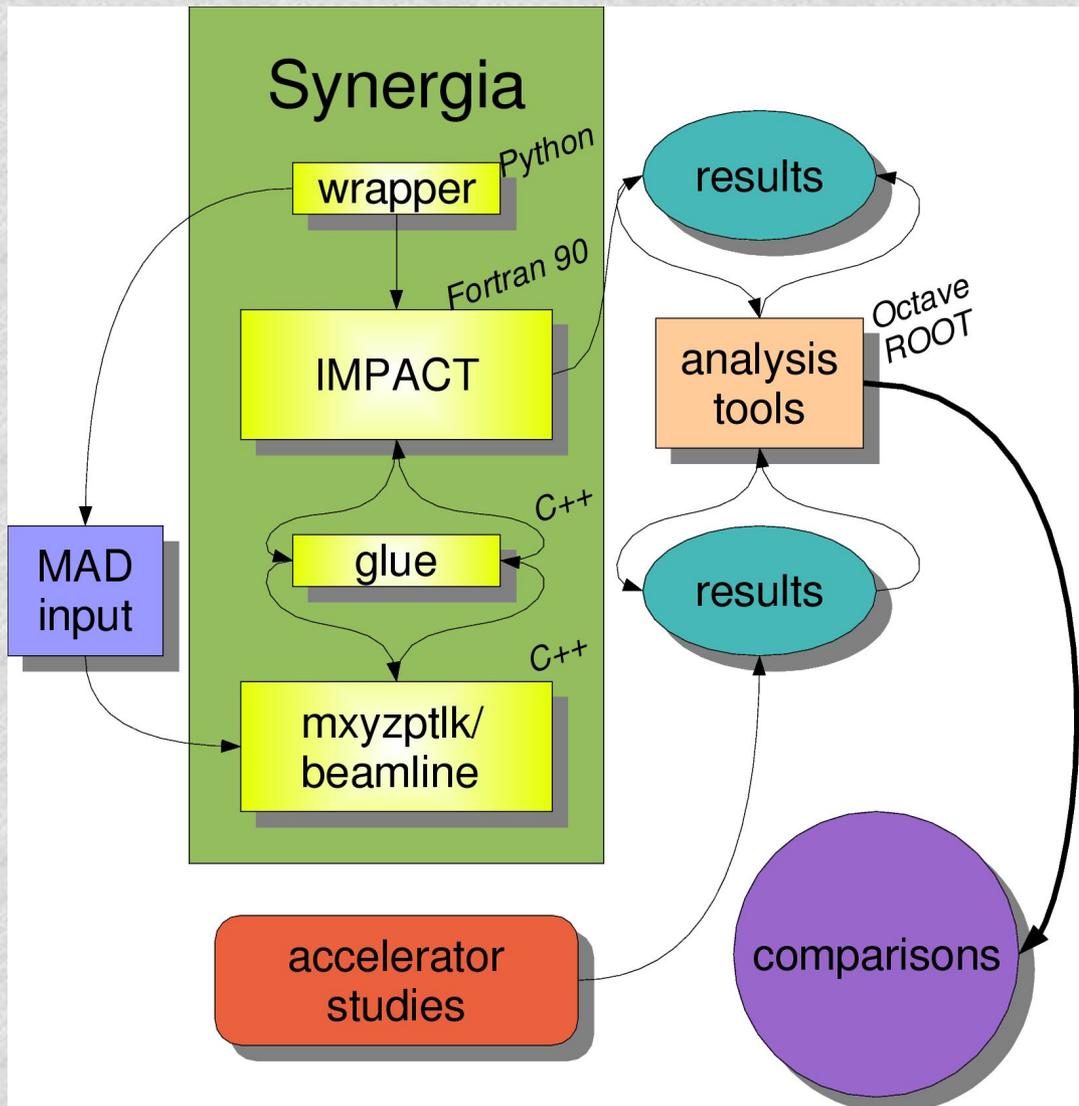


**UCLA**  
Parallel PIC Frameworks



**LANL**  
High order optics, beam expts, collisions, multi-language support, statistical methods

# Synergia Overview



- Synergia is a combination of **IMPACT**, **mxyzptlk/beamline**, glue code and a wrapper
- **IMPACT**
  - program flow
  - space charge
- **mxyzptlk/beamline**
  - mad parser
  - linear maps

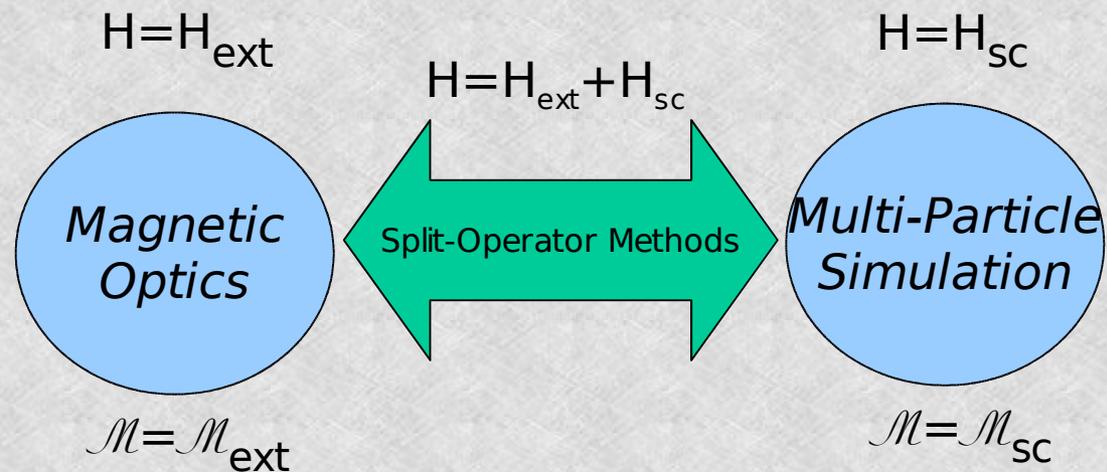
# IMPACT (Multi-particle code)

---

- Ji Qiang, Robert Ryne and Salman Habib
  - Developed at LANL, 2/3 now at LBNL
- Originally designed for linear accelerators
- Fortran 90
  - Fully parallel
    - Single-particle tracking
    - Space charge calculation
  - Full 3D space charge

# Space Charge in IMPACT

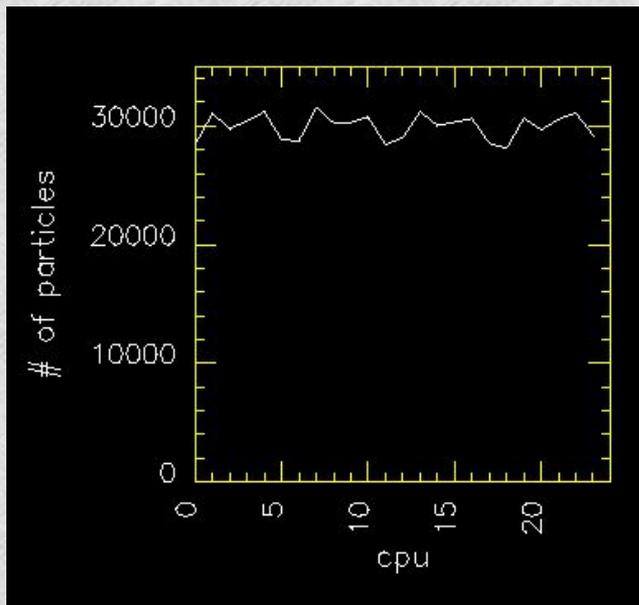
- Solve Poisson-Vlasov Equation
  - particle-in-cell (PIC)
- Split Operator Method



$$M(t) = M_{\text{ext}}(t/2) M_{\text{sc}}(t) M_{\text{ext}}(t/2) + O(t^3)$$

# Parallel Support in IMPACT

- Particle Manager
  - Distributes particles among processors
  - Re-distributes particles after they have moved
- Poisson-Vlasov solver
  - Distributes grid across all processors
  - Requires global communication



# mxyzptlk/beamline library suite (single-particle code)

---

- Leo Michelotti & Francois Ostiguy, FNAL
- C++
  - first C++ library for accelerator physics
- Flexible libraries
- Many features
  - Provides linear maps with arbitrary splittings
  - MAD file parser
    - Existing standard for lattice description
  - Much, much more...

# mxyzptlk/beamline libraries

---

- basic\_toolkit
  - Useful utility classes: Vector, Matrix...
- mxyzptlk
  - Automatic differentiation and differential algebra
- beamline
  - Objects for modeling elements of a beamline
- physics\_toolkit
  - analysis and computation
- Machines
  - FNAL models

# Language mixing

---

- Mixing F90 and C++ can be done
  - Requires some glue
  - Always platform/compiler-dependent
    - Macros for general code
    - Compiling and linking
      - see configuration management
- We have to be very careful to ensure that our hybrid code is portable to multiple platforms
  - ... particularly the platforms of interest
    - Including supercomputers with “*interesting*” characteristics

# Glue in Synergia

---

- Added a new element to IMPACT, “external”
  - F90 modifications to IMPACT
  - New skeleton F90 module
- Created a C++ class, External\_class, to get maps from a MAD file using beamline
- “F90 module” really C++ code that talks to External\_class

# Configuration Management

---

- Building and installing code is not glamorous
  - In a mixed-language environment it is also not trivial
- Configuration management can rapidly become a bottleneck
- GNU Autotools
  - “The worst possible choice, except for all the others”

# Building Synergia with GNU Autotools

---

- *First principle: no editing of source or build files should be necessary*
- Record known solutions
  - Not “the autotools way”
- In principle
  - Get mxyzptlk/beamline libraries
    - `./configure && make && make install`
  - Get IMPACT
    - `./configure && make`
- In practice, there are more decisions to make...

# In practice

```
> ./configure --help
<snip>
--with-glib-prefix=PFX      Prefix where GLIB is installed (optional)
--with-glib-exec-prefix=PFX Exec prefix where GLIB is installed (optional)
--with-mxyzptlk-prefix=<dir> Prefix directory for mxyzptlk.
    Default is to search /usr/local, \${HOME}, \${HOME}/opt, \${HOME}/mxyzptlk
--with-mpi-prefix=<dir>     Prefix directory for MPI.
--with-mpi-include-dir=<dir> MPI include directory.
    Default is -I\${MPI_PREFIX}/include
--with-mpi-ldflags=<flags>  LDFLAGS for linking with MPI.
--with-mpi-libs=<libs>      Libraries for linking with MPI.
--with-hdf5-prefix=<dir>    Prefix directory for HDF5.
```

Some influential environment variables:

CXX	C++ compiler command
CXXFLAGS	C++ compiler flags
LDFLAGS	linker flags, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir>
CPPFLAGS	C/C++ preprocessor flags, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir>
F77	Fortran 77 compiler command
F77FLAGS	Fortran 77 compiler flags
CC	C compiler command
CFLAGS	C compiler flags

# Synergia Build System

---

- Default choices are stored in installation scripts (`configure.in`)
  - This is the “not the autotools way” part
- Compile-time choices are recorded (`config.status`)
- No source-code changes needed for configuration choices (`preprocessor macros`)
- New platforms are anticipated (`Autotools`)

# Synergia Interface

---

- The program flow in Synergia is controlled by IMPACT
  - IMPACT's interface is something less than human-friendly
    - See next slide
- Synergia wraps IMPACT with a Python Layer
  - Unit conversions
  - Defaults
  - Full power of Python
  - Job creation/ submission
    - Extensive features

# IMPACT Interface Example

```

16 4
6 2746880 1 0 1
65 65 65 4 0.04 0.04 1.51692
2 0 0
0.00268186, 0.000106579, 0 1.000000 1.000000 0.000000 0.000000
0.00268186, 0.000106579, 0 1.000000 1.000000 0.000000 0.000000
0.0940268, 0.000427895, 0 1.000000 1.000000 0.000000 0.000000
0.042 4e+08 9.38272e+08 1.000000 2.01e+08 0.000000
0 0 81 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 83 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 84 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 85 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 86 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 87 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 88 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 89 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 90 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 91 -2 270.000000/
474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
0 0 92 -2 270.000000/
0 0 82 -2 270.000000/

```

Computational parameters

Input distribution  
(in IMPACT's private  
system of units)

Lattice description

# Synergia Interface Example

```

myopts = options.Options("booster_test3")
myopts.add("numturns",10,"Number of turns",int)
myopts.add("xwidth",0.004,"horizontal width (m)",float)
myopts.add("dpop",3.0e-4,"(delta p)/p",float)

myopts.add_suboptions(impact_parameters.options)
myopts.add_suboptions(synergia.options)
myopts.parse_argv(sys.argv)

ip = impact_parameters.Impact_parameters(impact_parameters.options
                                         .get_value("sampleperiod"))
ip.apply_options(impact_parameters.options)
pz = ip.gamma() * ip.beta() * ip.mass_GeV
madfile = "booster_sliced.mad"
(D_x, D_y) = madcalc.dispersion_initial(madfile,"bcelinj",
                                       myopts.get_value("energy"))
(alpha_x, beta_x, alpha_y, beta_y) = madcalc.twiss_initial(madfile,
                                                           "bcelinj")

width_x = myopts.get_value("xwidth")
(width_xprime,r_x,emittance) =
    matching.match_twiss_width(width_x,alpha_x,beta_x)
ip.x_params(sigma = width_x, lam = width_xprime * pz)
booster = impact_elements.External_element(kicks=100, steps=1,
                                           radius=ipradius,
                                           mad_file_name=madfile)

numturns = myopts.get_value("numturns")
for turn in range(1,numturns+1):
    ip.add(booster)
    if turn != numturns:
        ip.add(impact_elements.Output_element("turn%02d.dat" % turn,
                                             sample_period))

```

User options and  
prepackaged options

Direct access to  
lattice functions  
from MAD files

Obvious names,  
physical units

Full access to python

# How Synergia manages job complexity

- Separates descriptions
  - MAD files for lattices
  - Python script for job
- Full power of Python
  - Easier to use
  - Easier to maintain
- Manages history and details

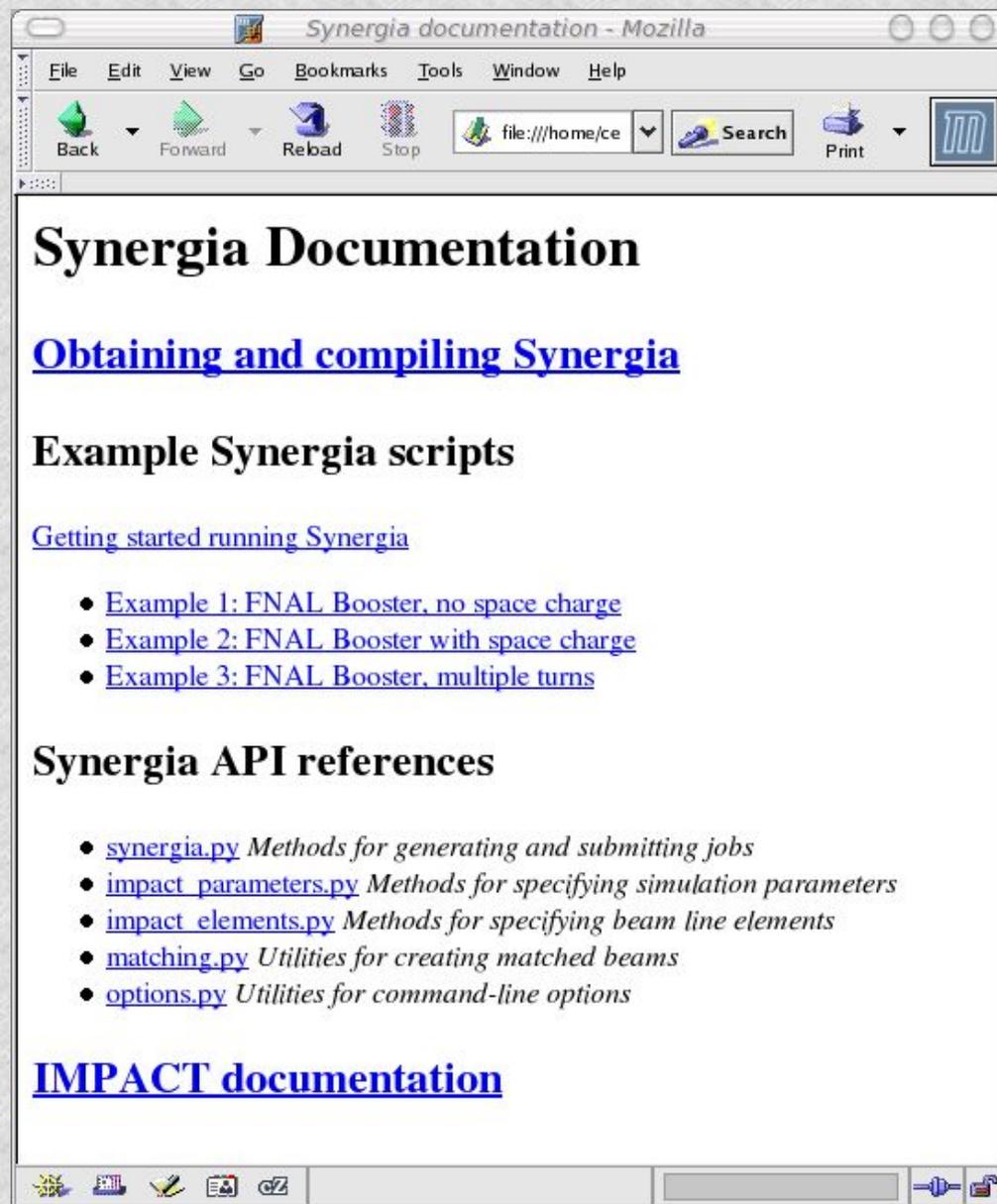
Accelerator  
simulation  
problems are  
not simply  
numerical

Job directory includes **batch script**, **utilities** and **history**:

<code>booster_test3.py*</code>	<code>description</code>	<code>pack*</code>	<code>wrappedxmain*</code>
<code>cleanup*</code>	<code>external_ble.mad@</code>	<code>synergia-pbs.sh*</code>	<code>xmain@</code>
<code>command</code>	<code>mad_mapping.table</code>	<code>test.in</code>	

# Documentation

- Documentation is in the form of web pages
  - Examples
  - API
- New external users are starting to help us refine the documentation
- First applications
  - FNAL Booster
  - A0 Photoinjector



# API Documentation

## matching

[index](#)</home/amundson/work/beams/Layer/matching.py>

### Functions

**match\_twiss\_emittance**(emittance, alpha, beta)

Calculate input parameters for a matched beam of given width using Courant-Snyder (Twiss) parameters. Returns

```
(width,width_prime,r),
```

where width and width\_prime are the width in the coordinate and its conjugate and r is the correlation coefficient.

**match\_twiss\_width**(width, alpha, beta)

Calculate input parameters for a matched beam of given width using Courant-Snyder (Twiss) parameters. Returns

```
(width_prime,r,emittance),
```

where width\_prime is the width in the conjugate coordinate and r is the correlation coefficient.

- Documentation is automatically generated from source
  - always accurate
  - extensible
  - takes advantage of the power of Python

# API Documentation, cont.

- Methods have descriptive names
  - optional text
- Variables have descriptive names
  - including units

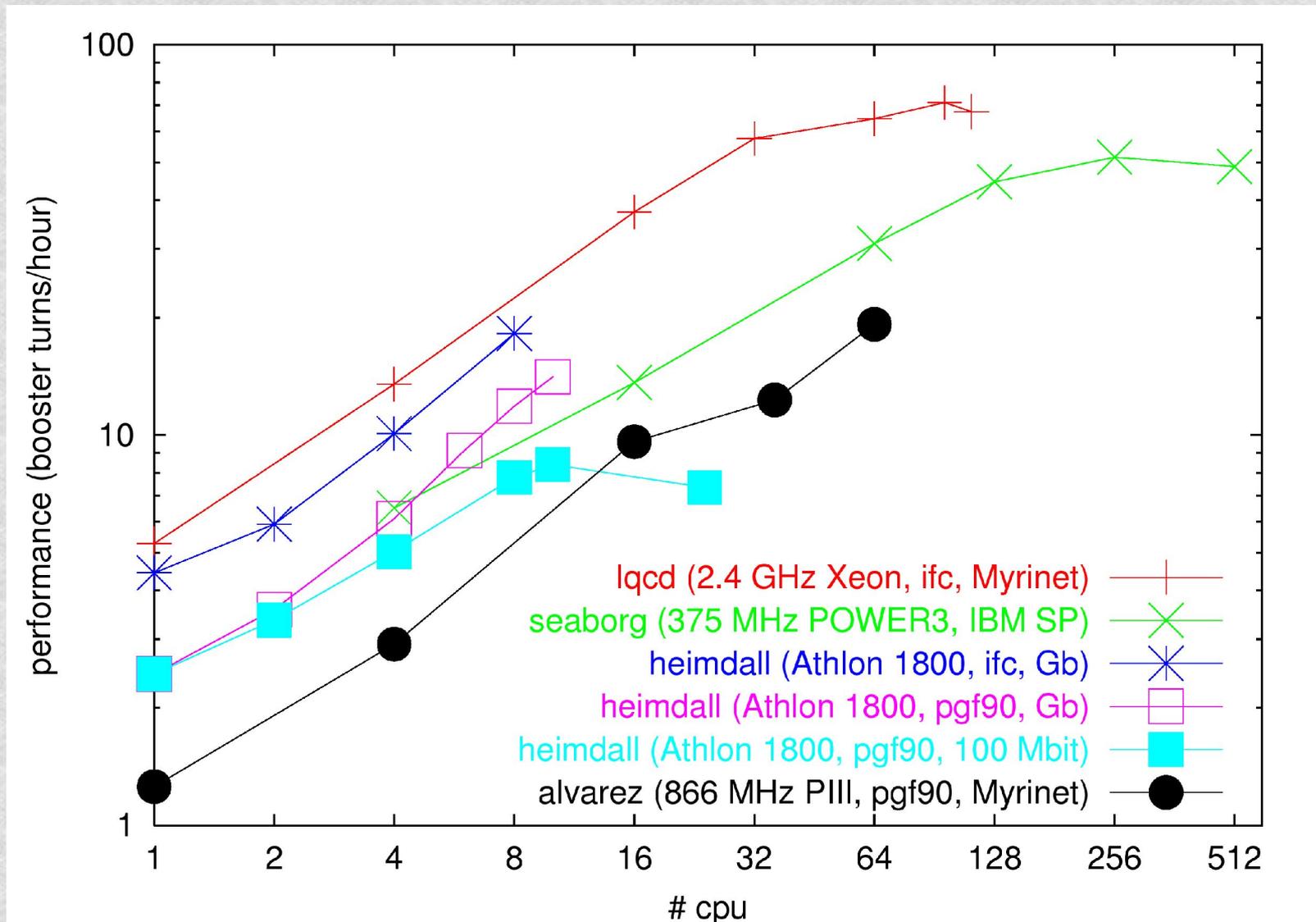
```
class Impact_parameters
```

```
    __init__(self, sampling_period=1)
    _adjust_particles(self)
    add(self, element, final=0)
    beta(self)
    → returns the velocity of the reference particle in natural units
    charge(self, charge_e)
    current(self, current_A)
    describe(self, file=<open file '<stdout>', mode 'w'>)
    dimensions(self, dim)
    error_studies(self, bool)
    gamma(self)
    initial_phase(self, phase)
    input_distribution(self, name)
    integrator(self, name)
    kinetic_energy(self, energy_GeV)
    mass(self, mass_GeV)
    omega(self)
    output_type(self, name)
    particles(self, number)
    pipe_dimensions(self, width_x_m, width_y_m)
    processors(self, procs_per_node, nodes)
    restart(self, bool)
    scaling_frequency(self, frequency_Hz)
    space_charge_BC(self, name)
    space_charge_grid(self, nx, ny, nz)
```

# Performance and Parallelism

- Ran benchmarks on five machines
  - **abacus**
    - my 800 MHz laptop
  - **heimdall (FNAL)**
    - 32 dual 1.4 GHz Athlons, 100 Mbit/s and Gigabit interfaces
  - **alvarez (NERSC)**
    - 80 dual 866 MHz PIII, Myrinet
  - **Seaborg (NERSC)**
    - 6,000+ 375 MHz POWER3 processors
  - **Lattice QCD (FNAL)**
    - 128 dual 2.4 GHz Xeons, Myrinet

# Results with space charge

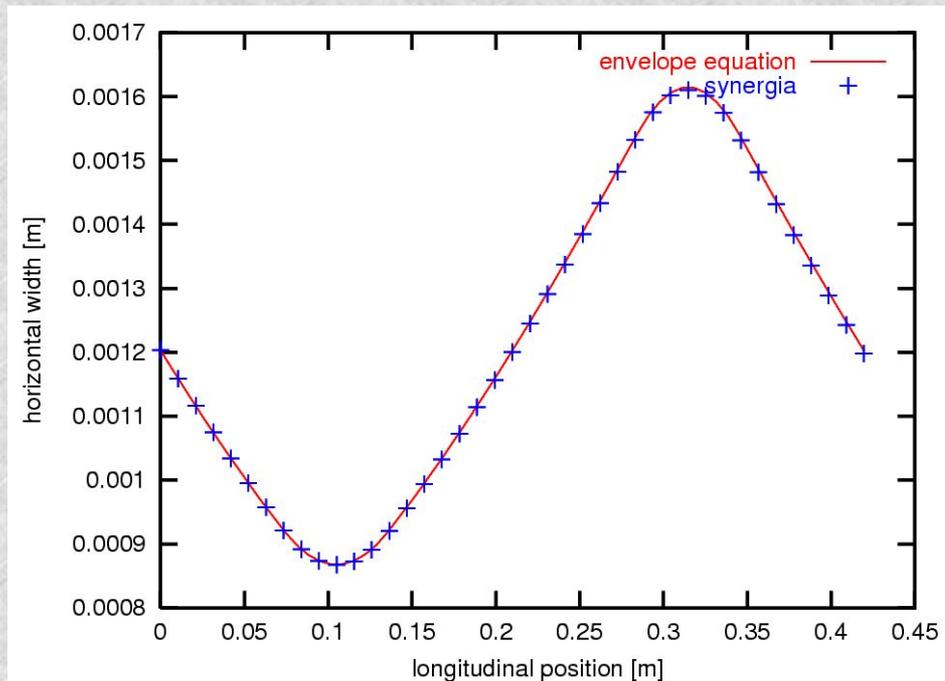


# But... are the answers correct?

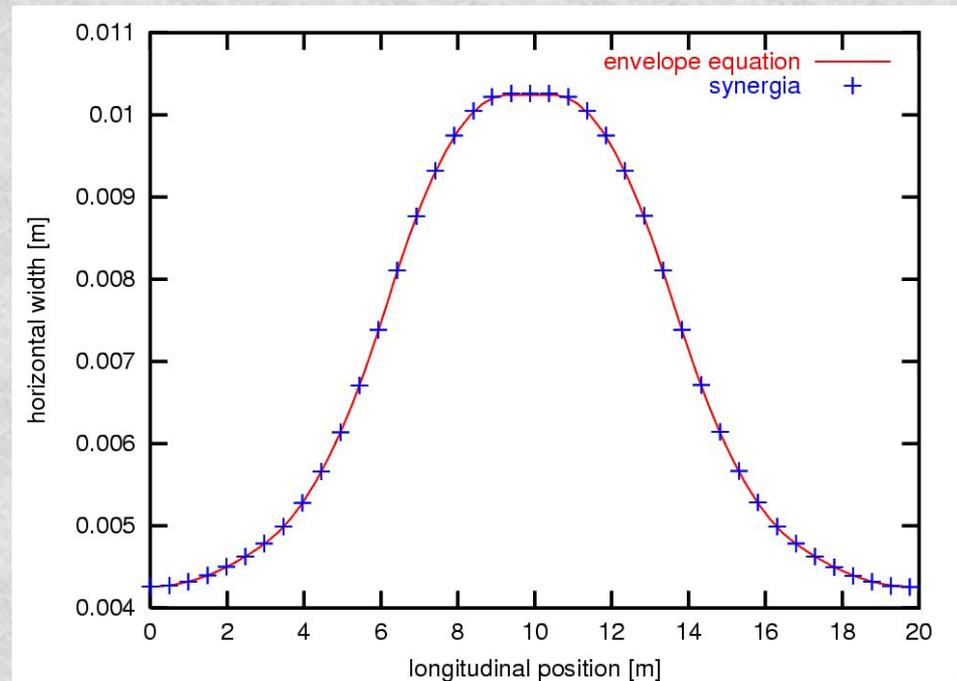
## Validation

- Envelope equations: coupled differential equations describing the evolution of the 2<sup>nd</sup> moments of the beam distribution including space charge

### FODO channel, KV beam



### Booster cell, Gaussian beam



# Particle Manager Performance

---

- An example of a known issue
- Terminology
  - Matched beam
    - Periodically returns to same state
  - Mismatched beam
    - Rotates in phase space
    - Oscillates in width
- Mismatched beam can lead to a factor of 3 disparity in distribution of particles among processors
- (see particle manager visualization)

# Next steps in development

---

- Synergia development to date has been driven by FNAL Booster applications
- Further applications will be easier if the program flow is made more flexible
  - Adding new physics can be made much easier
- This is work in progress

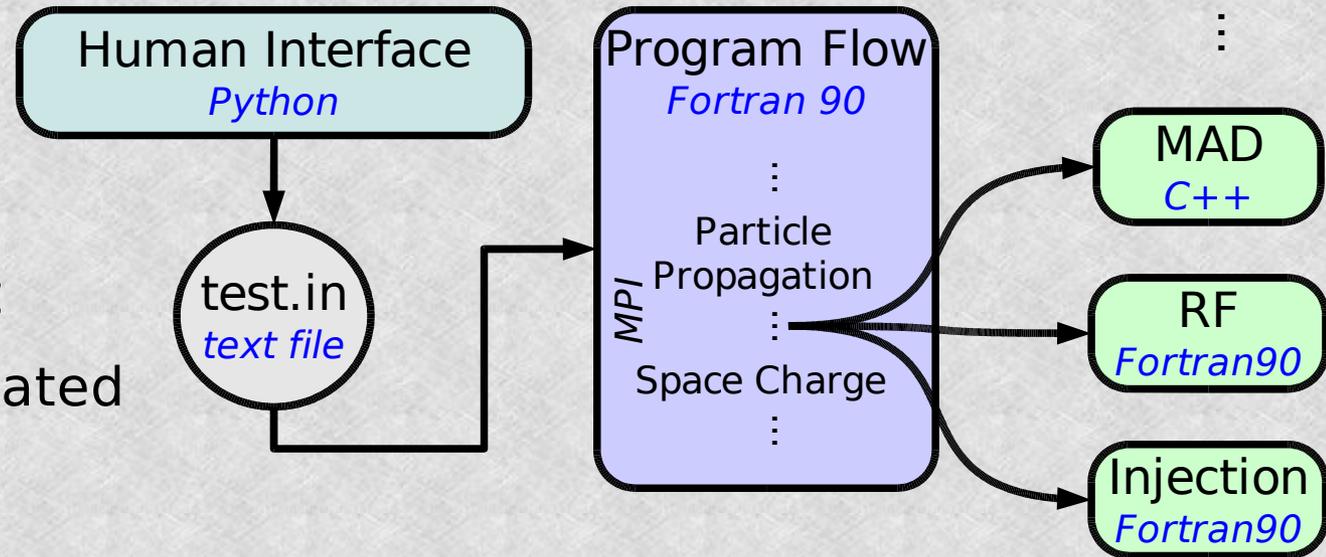
# Current Scheme

- *pros*

- simple to implement
- minimal platform-related problems

- *cons*

- difficult to extend for new physics
  - must extend IMPACT object system with  $N^2$  connections

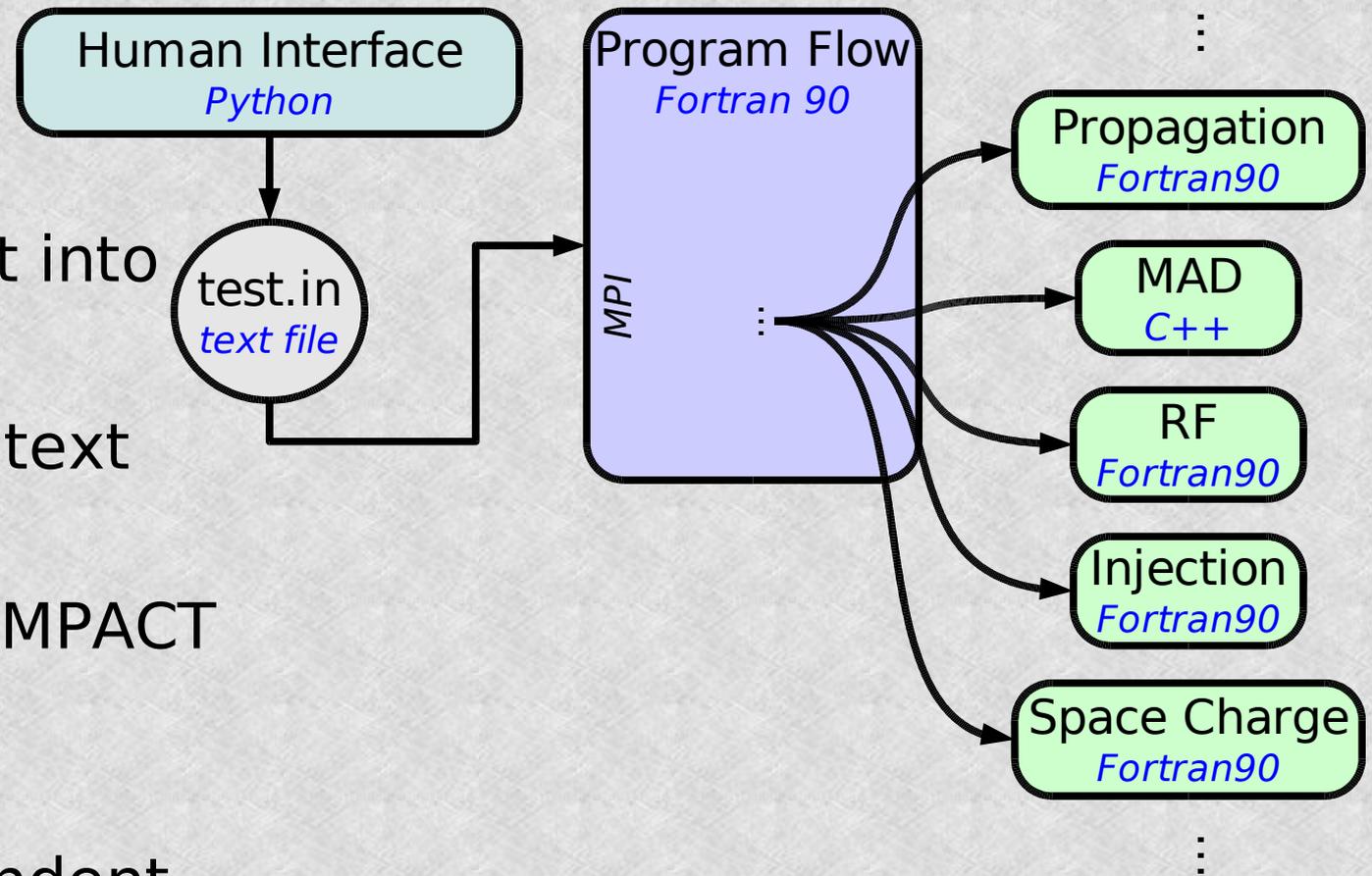


- *cons, cont.*

- interfaces pass through text file
  - ...and must be parsed by Fortran code
- IMPACT features (parallelism, SC) not exportable

# Next Step

- program flow split into modules
  - still limited by text file
  - still bound by IMPACT object system
  - no additional platform-dependent problems



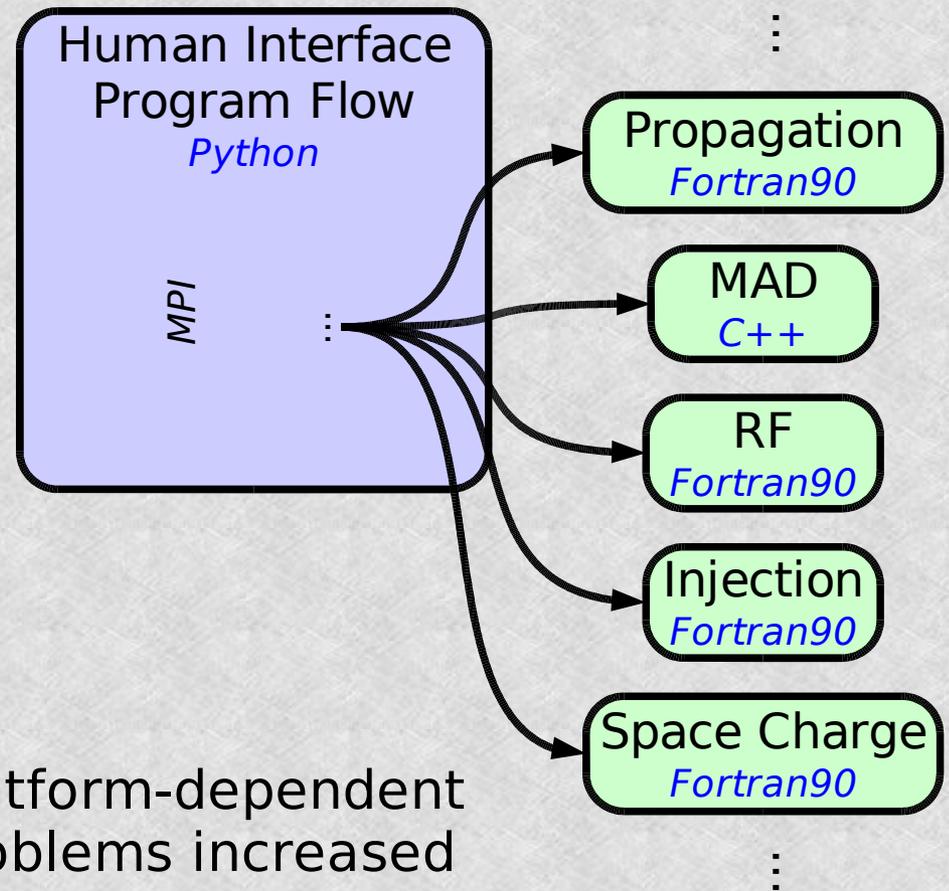
# Final Goal

- *pros*

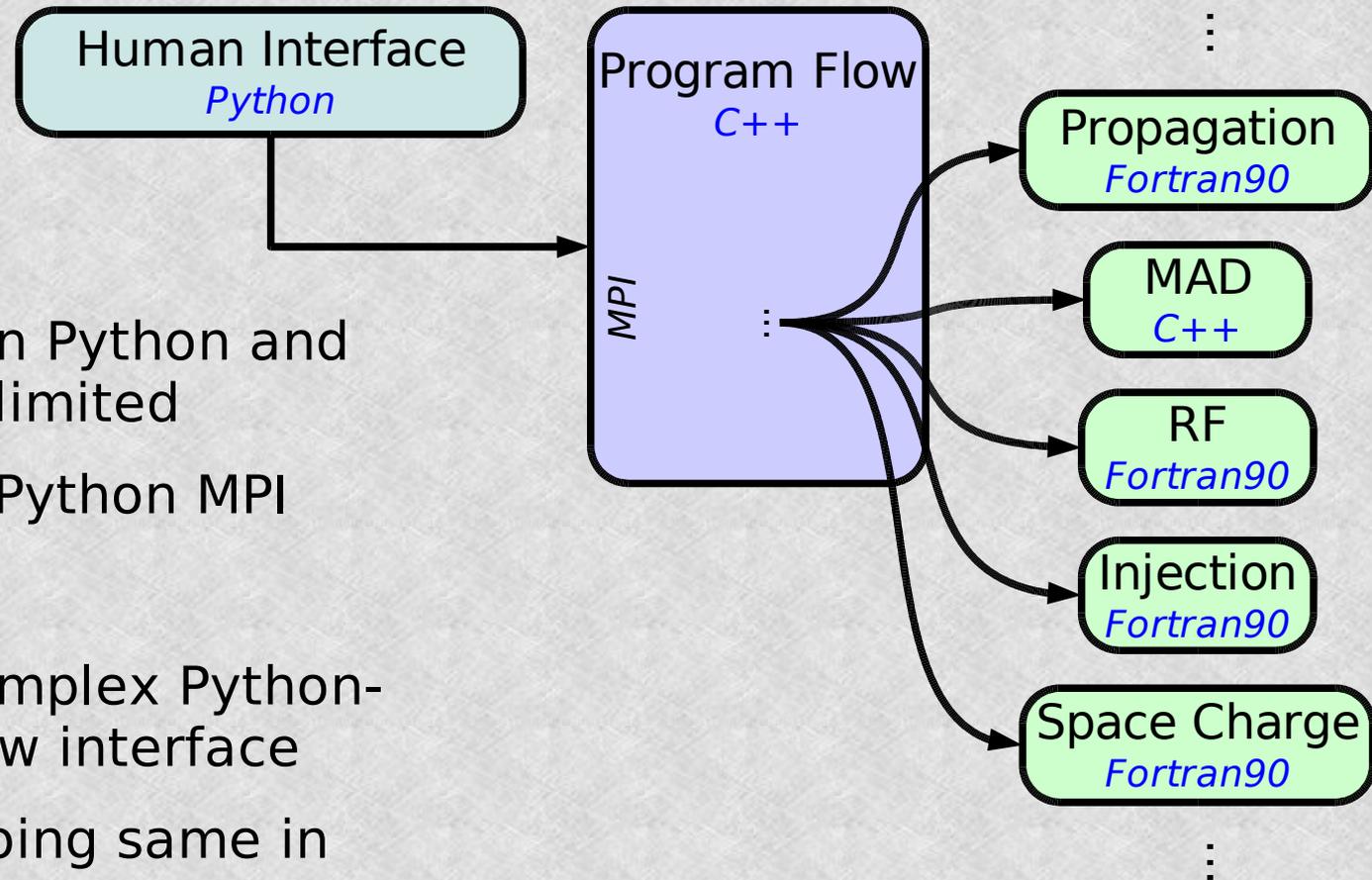
- greatly enhanced flexibility
  - new physics modules can be trivially added
    - beam-beam
    - etc.
  - dynamic loading possible
- better interface definition

- *cons*

- platform-dependent problems increased
  - Python MPI
  - Calls between Python, Fortran 90 and C++



# Alternate Plan



- *pros*

- interface between Python and other languages limited
- does not rely on Python MPI

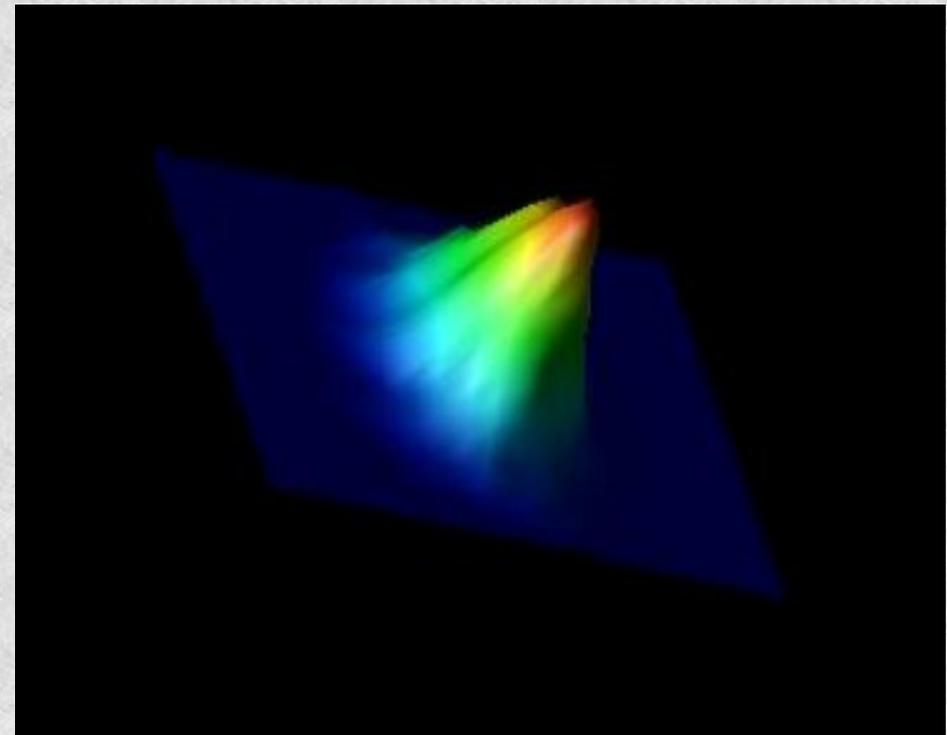
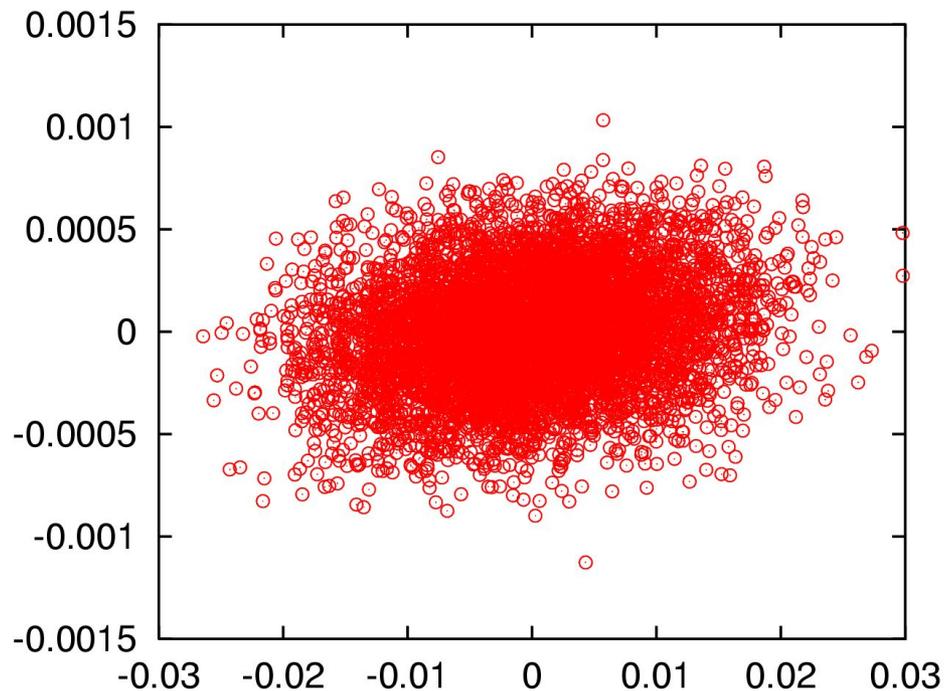
- *cons*

- requires more complex Python-C++ program flow interface
  - easier than doing same in Fortran90

# Visualization

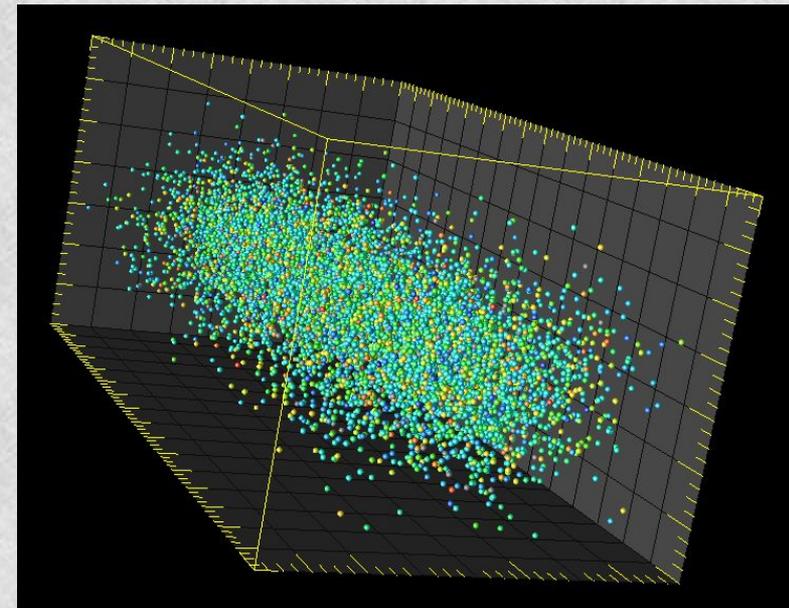
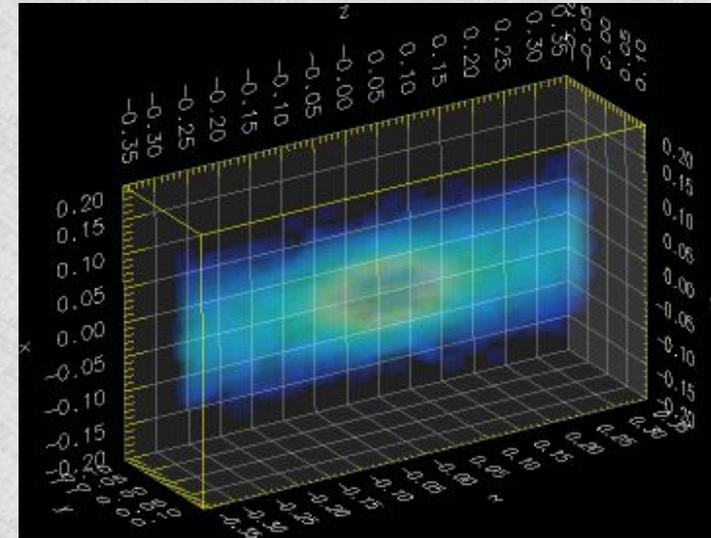
- Two-dimensional plotting tools are inadequate for analyzing data in many dimensions

Horizontal phase space ( $x, p_x$ )

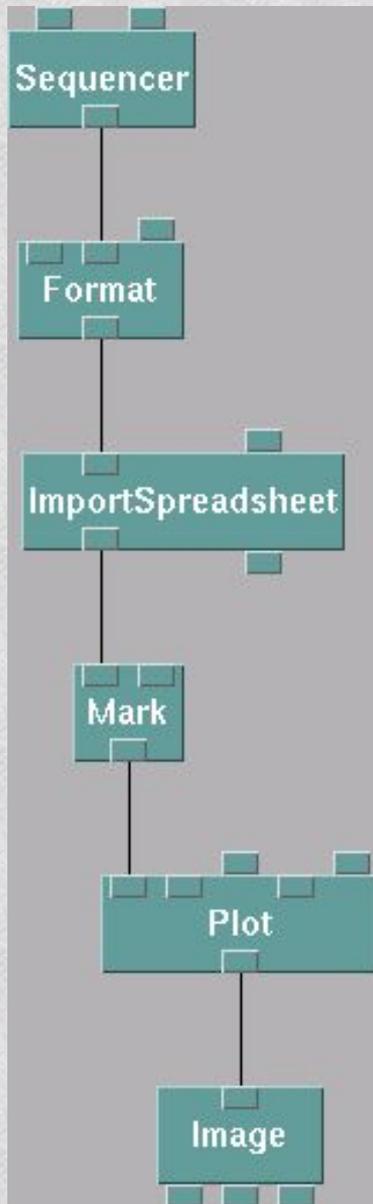


# OpenDX

- OpenDX is a multi-dimensional visualization package
  - Originally IBM's DX (\$\$\$\$)
  - Very powerful
  - Very steep initial learning curve
  - Visual programming
    - Also has a scripting language
  - Animation is easy



# OpenDX Example



Notation: `[Plot]`

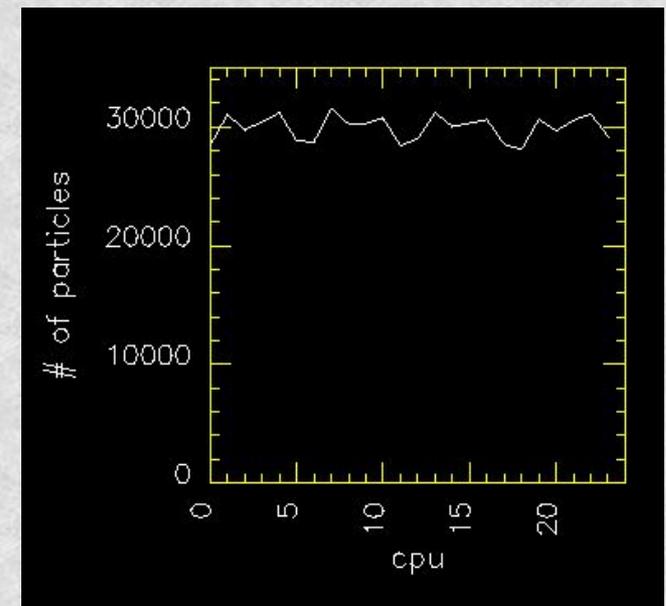
Inputs:

Name	Hide	Type	Source	Value
<input checked="" type="checkbox"/> input	<input type="checkbox"/>	field, group	Mark	NULL
<input checked="" type="checkbox"/> labels	<input type="checkbox"/>	string list		["cpu", "# of particles"]
<input type="checkbox"/> ticks	<input type="checkbox"/>	integer list		(input dependent)
<input checked="" type="checkbox"/> corners	<input type="checkbox"/>	vector list, object		[0,0],[24,35000]
<input type="checkbox"/> aspect	<input type="checkbox"/>	scalar, string		1.0

Outputs:

Name	Type	Destination	Cache
plot	field	Image	All Results <input type="checkbox"/>

Buttons: OK, Apply, Expand, Collapse, Description..., Help on Syntax, Restore, Cancel



# A more complex example

